

## lecture 5

Brad McNeney

2017-01-29

# Working with character strings

Fixed, or literal strings:

- ▶ count the number of characters in a string
- ▶ detect (yes/no) or find (starting position) substrings
- ▶ extract and substitute substrings
- ▶ split and combine strings

String patterns:

- ▶ detect, find, extract and substitute

Throughout, illustrate “base” R utilities and those from the `stringr` package

# The 'stringr' package

- ▶ Character string manipulation in R has evolved as a collection of tools whose name and function is taken from string manipulation tools in Unix and other scripting languages such as Perl.
- ▶ The `stringr` package aims for a cleaner interface for tasks that relate to detecting, extracting, replacing and splitting on substrings.

```
library(stringr)
```

## Counting the number of characters

```
mystrings <- c("one fish", "two fish", "red fish", "blue fish")  
nchar(mystrings)
```

```
## [1] 8 8 8 9
```

# Detecting substrings

- ▶ `grep()` returns the indices of strings that contain a substring, while `grepl()` returns a logical vector:

```
pattern <- "red"  
grep(pattern,mystrings)
```

```
## [1] 3
```

```
mystrings[grep(pattern,mystrings)]
```

```
## [1] "red fish"
```

```
grepl(pattern,mystrings)
```

```
## [1] FALSE FALSE TRUE FALSE
```

```
mystrings[grepl(pattern,mystrings)]
```

```
## [1] "red fish"
```

## Detecting substrings with `stringr::str_detect()`

- Works like `grep1()` but note that we *switch* the order of the arguments:

```
str_detect(mystrings,pattern)
```

```
## [1] FALSE FALSE  TRUE FALSE
```

```
mystrings[str_detect(mystrings,pattern)]
```

```
## [1] "red fish"
```

# Finding substring starting position

- `regexr()` returns the start of the first occurrence of a pattern, `gregexpr()` returns the start of all occurrences.

```
Seuss <- paste(mystrings,collapse=", "); Seuss
```

```
## [1] "one fish, two fish, red fish, blue fish"
```

```
regexr("fish",Seuss)
```

```
## [1] 5  
## attr("match.length")  
## [1] 4  
## attr("useBytes")  
## [1] TRUE
```

```
gregexpr("fish",Seuss)
```

```
## [[1]]  
## [1] 5 15 25 36  
## attr("match.length")  
## [1] 4 4 4 4  
## attr("useBytes")  
## [1] TRUE
```

## Finding substring starting position with stringr

- ▶ stringr analogs to regexpr and gregexpr are str\_locate and str\_locate\_all, with argument order reversed.

```
str_locate(Seuss, "fish")
```

```
##      start end  
## [1,]      5  8
```

```
str_locate_all(Seuss, "fish")
```

```
## [[1]]  
##      start end  
## [1,]      5  8  
## [2,]     15 18  
## [3,]     25 28  
## [4,]     36 39
```



## Extracting substrings by start and stop position

- ▶ We saw `substr()` in the example of lecture 3 where we read in purchase amounts and wanted to remove the \$.
- ▶ Takes a character string, or vector of strings, as argument. Specify start and stop character.
- ▶ Another example

```
substr("this string has 30 characters!",start=10,stop=20)
```

```
## [1] "ng has 30 c"
```

## substr() with big start and stop

```
bignum <- 1000  
substr("this string has 30 characters!",start=10,stop=bignum)
```

```
## [1] "ng has 30 characters!"
```

```
substr("this string has 30 characters!",start=31,stop=bignum)
```

```
## [1] ""
```

- ▶ If stop greater than number of characters, stop at the end of the string.
- ▶ If start greater than number of characters, return ""

## Note: substr can do replacements to character variables

But its use is not very intuitive:

```
x<-"this string has 30 characters!"  
substr(x,start=10,stop=20)  <- c("X")  
x
```

```
## [1] "this striXg has 30 characters!"
```

```
substr(x,start=10,stop=20)  <- c("XXXXXXX")  
x
```

```
## [1] "this striXXXXXXXX30 characters!"
```

```
substr(x,start=10,stop=20)  <- c("XXXXXXXXXXXXXXXXXX")  
x
```

```
## [1] "this striXXXXXXXXXXXXharacters!"
```

## Replacing (substituting) substrings

- ▶ `sub()` and `gsub()` replace the first and all occurrences of a substring with a replacement.

```
sub("fish", "bird", Seuss)
```

```
## [1] "one bird, two fish, red fish, blue fish"
```

```
gsub("fish", "bird", Seuss)
```

```
## [1] "one bird, two bird, red bird, blue bird"
```

## Replacing substrings with stringr

- Use `str_replace` and `str_replace_all`.

```
str_replace(Seuss, "fish", "bird") # replace first occurrence
```

```
## [1] "one bird, two fish, red fish, blue fish"
```

```
str_replace_all(Seuss, "fish", "bird") # replace all
```

```
## [1] "one bird, two bird, red bird, blue bird"
```

## Splitting strings with strsplit

- `strsplit()` splits a vector of character strings on a specified separator and returns a list.

```
mystrings <- c("this is a string", "so is this")  
strsplit(mystrings,split=" ")
```

```
## [[1]]  
## [1] "this"    "is"      "a"       "string"  
##  
## [[2]]  
## [1] "so"     "is"     "this"
```

## strsplit() on special characters

- ▶ Some characters, such as ., have special meaning when used as part of the split argument.
  - ▶ more on these special characters and “regular expressions” soon
- ▶ To match the split argument exactly, use fixed=TRUE

```
mystrings <- c("20.50", "33.33")  
strsplit(mystrings,split=".")
```

```
## [[1]]  
## [1] "" "" "" "" ""  
##  
## [[2]]  
## [1] "" "" "" "" ""
```

```
strsplit(mystrings,split=".",fixed=TRUE)
```

```
## [[1]]  
## [1] "20" "50"  
##  
## [[2]]  
## [1] "33" "33"
```

## Splitting with stringr

- ▶ The `str_split()` command is similar to `strsplit()`, but with argument `pattern` instead of `split`.
  - ▶ wrap pattern in `fixed()` for a fixed string

```
str_split(mystrings,pattern=".")
```

```
## [[1]]  
## [1] "" "" "" "" "" ""  
##  
## [[2]]  
## [1] "" "" "" "" "" ""
```

```
str_split(mystrings,pattern=fixed("."))
```

```
## [[1]]  
## [1] "20" "50"  
##  
## [[2]]  
## [1] "33" "33"
```



## Combining strings with paste()

- ▶ `paste()` glues together strings or vectors of strings separated by a user-specified separator (default " ").
  - ▶ The default separator of `paste0()` is no-space "".

```
mystrings <- c("21.33", "33.33")  
paste(mystrings[1], mystrings[2])
```

```
## [1] "21.33 33.33"
```

```
paste("$", mystrings, sep="")
```

```
## [1] "$21.33" "$33.33"
```

- ▶ We can also paste together elements of a vector

```
paste(mystrings, collapse=" ")
```

```
## [1] "21.33 33.33"
```

## Working with string patterns: regular expressions

- ▶ The string manipulations so far that involve substrings have used fixed, or literal, substrings.
- ▶ Sometimes we would prefer to identify strings that match a pattern.
- ▶ A regular expression (abbreviated regex) is a string of characters used to specify a search pattern
- ▶ Regular expressions is a complex topic. We'll only cover a simple case.
- ▶ Learn more with the following references:
  - ▶ RStudio Regular Expressions Cheatsheet:  
<https://www.rstudio.com/wp-content/uploads/2016/09/RegexCheatsheet.pdf>
  - ▶ Regular expressions section of Prof. Bryan's Stat545 at UBC  
[http://stat545.com/block028\\_character-data.html](http://stat545.com/block028_character-data.html)
  - ▶ The Strings chapter of R for Data Science  
<http://r4ds.had.co.nz/strings.html>

## A simple pattern with .

- ▶ To illustrate pattern matching, use a simple pattern `p.n`, meaning `p` followed by any any character, followed by `n`.

```
pattern <- "p.n"  
mystrings <- c("pineapple", "apple", "pen")
```

# Detecting patterns

- ▶ The functions `grep`, `grep1` and `str_detect` all accept regular expressions as the pattern to find; e.g.,

```
str_detect(mystrings,pattern)
```

```
## [1] TRUE FALSE TRUE
```

## Splitting on a pattern

- `strsplit` and `str_split` accept regular expressions to split on; e.g.,

```
str_split(mystrings,pattern)
```

```
## [[1]]  
## [1] ""      "eapple"  
##  
## [[2]]  
## [1] "apple"  
##  
## [[3]]  
## [1] ""  ""
```

## Locating a pattern

- ▶ The string location functions `regexpr`, `gregexpr`, `str_locate` and `str_locate_all` accept regular expressions; e.g.,

```
str_locate(mystrings,pattern)
```

```
##      start end
## [1,]     1   3
## [2,]    NA  NA
## [3,]     1   3
```

# Extracting patterns

- ▶ We previously extracted substrings based on start and stop position.
- ▶ Can also extract patterns.

```
str_extract(mystrings,pattern)
```

```
## [1] "pin" NA      "pen"
```

```
str_match(mystrings,pattern)
```

```
##      [,1]  
## [1,] "pin"  
## [2,] NA  
## [3,] "pen"
```

# Replacing patterns

- ▶ `sub`, `gsub`, `str_replace` and `str_replace_all` accept regular expressions; e.g.,

```
str_replace(mystrings,pattern,"PPAP")
```

```
## [1] "PPAPeapple" "apple"      "PPAP"
```

- ▶ The replacement string is literal; e.g.,

```
str_replace(mystrings,pattern,"p.n")
```

```
## [1] "p.neapple" "apple"      "p.n"
```



## Adding \* and + quantifiers to .

- ▶ The combinations .\* and .+ match multiple characters.
  - ▶ E.G., f.\*t matches f followed by 0 or more characters, followed by t.
  - ▶ f.+t matches f followed by 1 or more characters, followed by t.

```
mystrings <- c("file1.txt","file10.txt","filetxt")  
pattern1 <- "file.*txt"; pattern2 <- "file.+txt"  
str_extract(mystrings,pattern1)
```

```
## [1] "file1.txt" "file10.txt" "filetxt"
```

```
str_extract(mystrings,pattern2)
```

```
## [1] "file1.txt" "file10.txt" NA
```